Implementation of CG Method on GPU Cluster with Proprietary Interconnect TCA for GPU Direct Communication

Kazuya MATSUMOTO[†1], Toshihiro HANAWA[†2], Yuetsu KODAMA[†4], Hisafumi FUJII[†5], Taisuke BOKU[†1,3]

†1 Center for Computational Sciences, University of Tsukuba

†2 Information Technology Center, The University of Tokyo

†3 Graduate School of Systems and Information Engineering, University of Tsukuba

†4 RIKEN Advanced Institute for Computational Science

†5 FUJITSU Software Technologies Limited

AsHES 2015

May 25, 2015

Outline

- Background and Motivation
- TCA (Tightly Coupled Accelerators) Architecture
- Collective Communication
  - Allgather and Allreduce
- CG Method
- Conclusion

Background

- GPU clusters are common as HPC systems
  - High peak performance / cost ratio
  - High peak performance / power ratio
- **Strong scaling** on GPU clusters is difficult.
  - Large gap between computation perf. and communication perf.
  - Communication latency between GPUs is larger than between CPUs
- Improving communication performance between GPUs is demanded for HPC
  - Our target is to develop a direct communication system between GPUs over different nodes for future accelerated computing
    ⇒ Tightly Coupled Accelerators (TCA) architecture

Our Previous Work on TCA

1. "**Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators**," In *AsHES 2013*.
   - Introduction (descriptions) on the TCA architecture
   - Performance evaluation on the ping-pong communication of TCA
2. "**QCD Library for GPU Cluster with Proprietary Interconnect for GPU Direct Communication**," In *HeteroPar 2014*.
   - Application of TCA to improve the communication performance in QUDA QCD library

Motivation

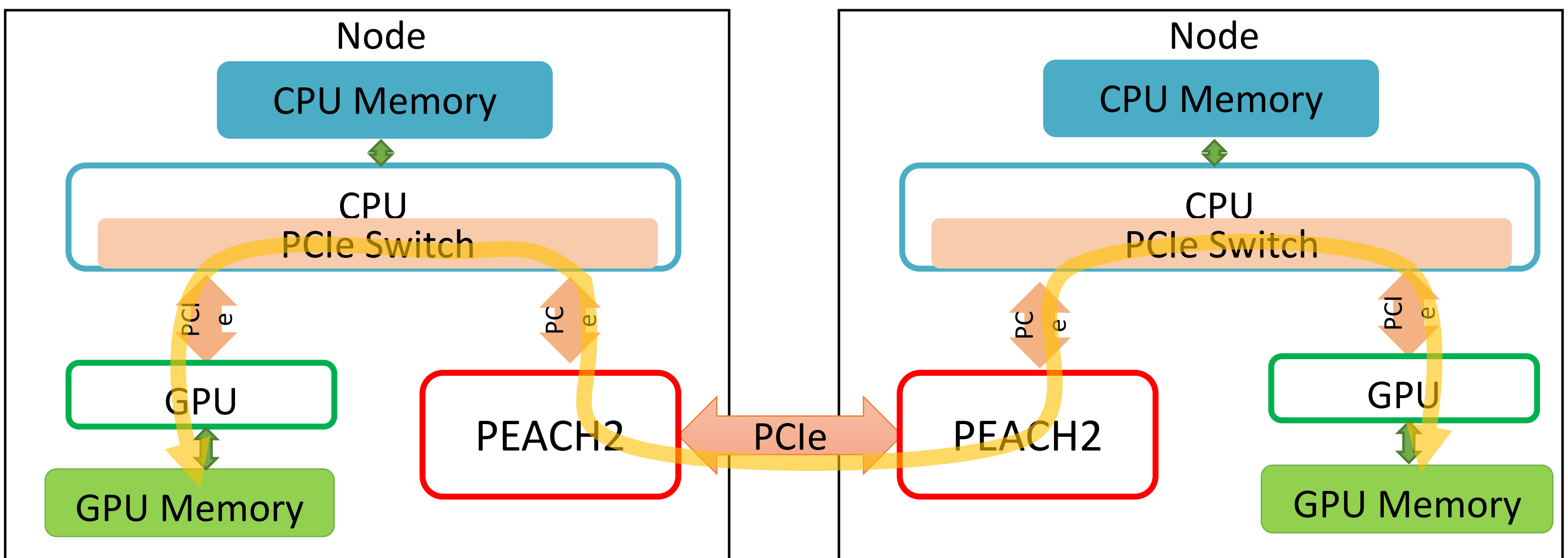
- Further performance evaluation of TCA
- Implementing **CG method** by using TCA
  - CG method: Iterative solution for systems of linear equations
  - Implementing **allgather and allreduce collective communication** with TCA API
- Evaluating the performance and seeing how TCA is effective

Outline

- Background and Motivation
- **TCA (Tightly Coupled Accelerators) Architecture**
- Collective Communication
  - Allgather and Allreduce
- CG Method
- Conclusion

TCA (Tightly Coupled Accelerators) Architecture

- Technology for direct connection between accelerators (GPUs) over different nodes without CPU assistance.
  - Low communication latency
    - By eliminating extra data copy to the host (CPU)
    - Improves strong scalability



PEACH2

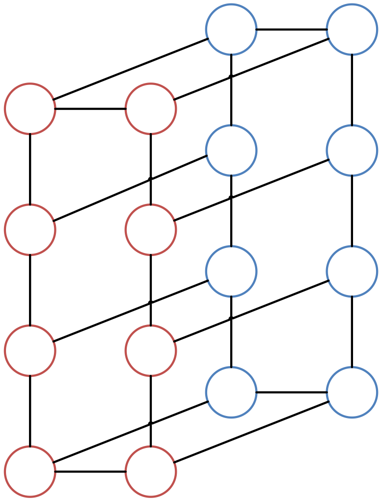
- PCI Express Adaptive Communication Hub ver. 2
- Implementation of TCA by FPGA
- Enables direct connection between GPUs with **PCI-Express (PCIe)** technology
  - Direct data copy is accomplished by NVIDIA GPUDirect Support for RDMA (GDR)
  - Protocol conversion is not required

    ⇒ Lower latency than InfiniBand
- Contains 4 PCIe ports (3 external ports)
  - Each port has **PCIe Gen2 x8** bandwidth (4 GB/s peak)
- NOTE: For convenience, we call this implementation of TCA on PEACH2 as "*TCA*".

HA-PACS/TCA

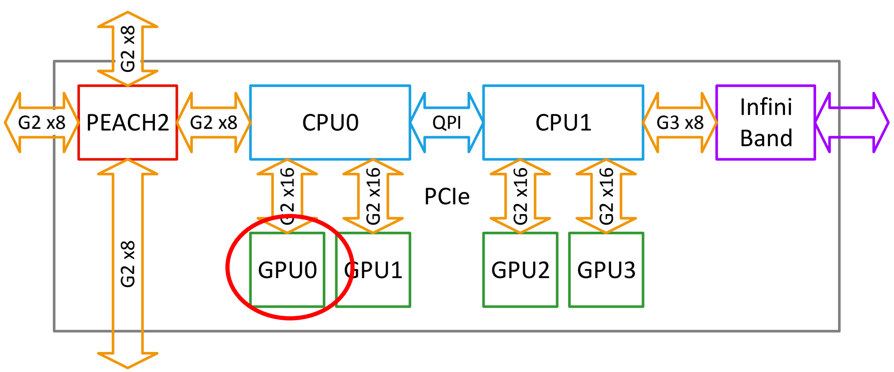
- Proof-of-concept GPU cluster of TCA concept in HA-PACS project

- 64 compute nodes in total
  - 4 sub-clusters each of which consists of 16 nodes
  - PEACH2 is equipped with
    - Sub-cluster configures 2x8 ring (torus) network.
      - By connecting 3 neighbor nodes through 3 PCIe ports of PEACH2
  - MPI communication through InfiniBand is also possible.
    - Can be considered to be a normal GPU cluster
    - Full-bisection bandwidth fat-tree network.



Performance Evaluation Condition

- Evaluation on a sub-cluster of HA-PACS/TCA
  - Up to 16 nodes (processes)
  - Using 1 GPU / node



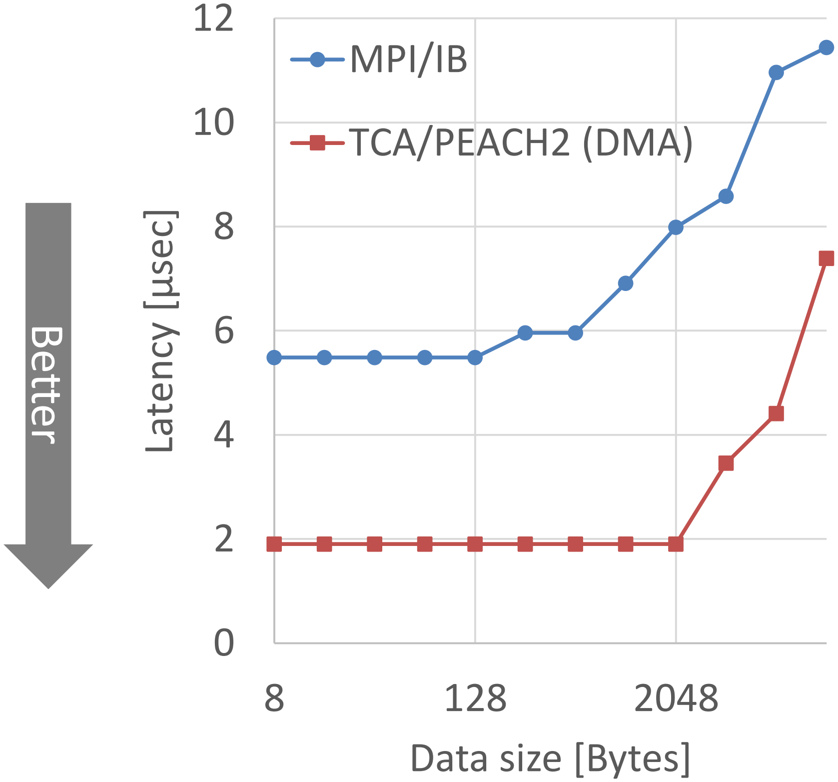
Hardware	
CPU	Intel Xeon E5-2680 2.8 GHz × 2 (**IvyBridge** 10 cores / CPU)
GPU	NVIDIA Tesla K20X × 4 (**Kepler GK110** 2688 cores / GPU)
TCA	**PEACH2 board** (Altera Stratix-IV GX 530 FPGA)
InfiniBand	Mellanox Connect-X3 Dual-port QDR
Software	
CUDA	6.5
MPI	MVAPICH 2 GDR 2.1a
C Compiler	Intel Compiler 14.0.3

MPI (MVAPICH2-GDR)

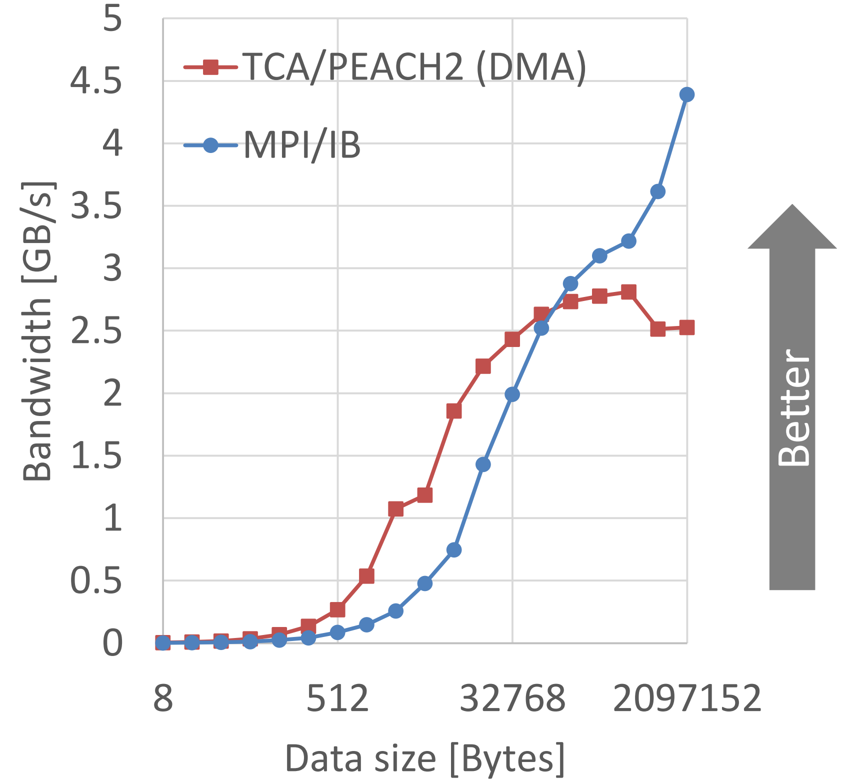
- We compare the performance of implementation using TCA with using MPI communication.
- MPI Impl.: **MVAPICH2-GDR 2.1a（MV2GDR)**
  - MPI implementation for InfiniBand
  - As with TCA, MV2GDR utilizes GPU Direct for RMA (GDR) to improve latency and bandwidth for small data communication

Ping-pong GPU-to-GPU Communication Performance

Latency



Bandwidth



- TCA/PEACH2 is better for small sizes.
- For large sizes, TCA is outperformed by MPI/IB since the difference of peak bandwidth perf. (4 GB/s vs. 8 GB/s) → **How about collective communications?**

Outline

- Background and Motivation
- TCA (Tightly Coupled Accelerators) Architecture
- **Collective Communication**
  - **Allgather and Allreduce**
- CG Method
- Conclusion

TCA Implementation of Collective Communication

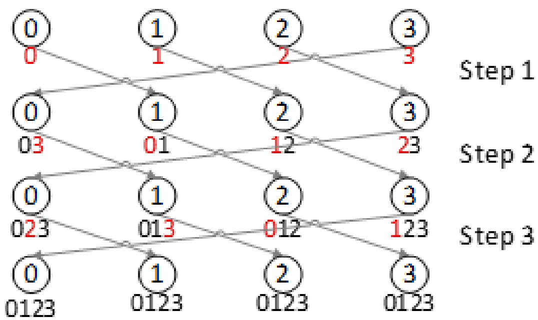
- **Allgather**
  - All processes gather data of each process.
  - Gathering data of KB-MB order
    - Communication bandwidth as well as latency is important.
- **Allreduce**
  - Conducts specified operation (sum, max, …) among data arrays (x_i) of each process and store the reduction result in all processes.
  - Targeted for CG method, we implement and tune allreduce (sum) for double-precision scalar (8 Bytes) data.
    - $(x_0 + x_1 + x_2 + x_3 = \sum_{i=0}^{3} x_i)$
    - Latency decides the performance.

Algorithms for Collective Communication

- Implement and evaluate 4 algorithms

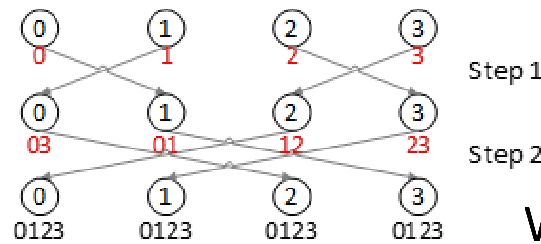
1. Ring algo.

Communication steps: p-1



3. Recursive Doubling algo.

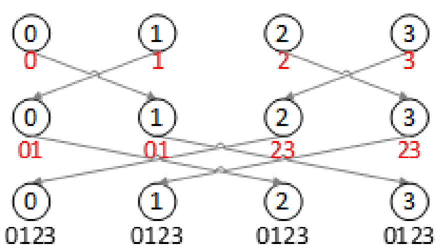
Communication steps: $\log_2 p$



We suppose #processes (p) is in power of 2.

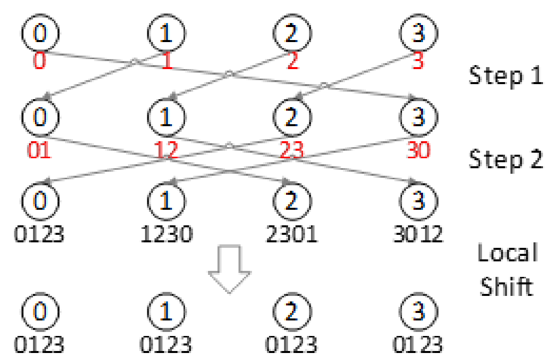
2. Neighbor Exchange algo.

Communication steps: p/2



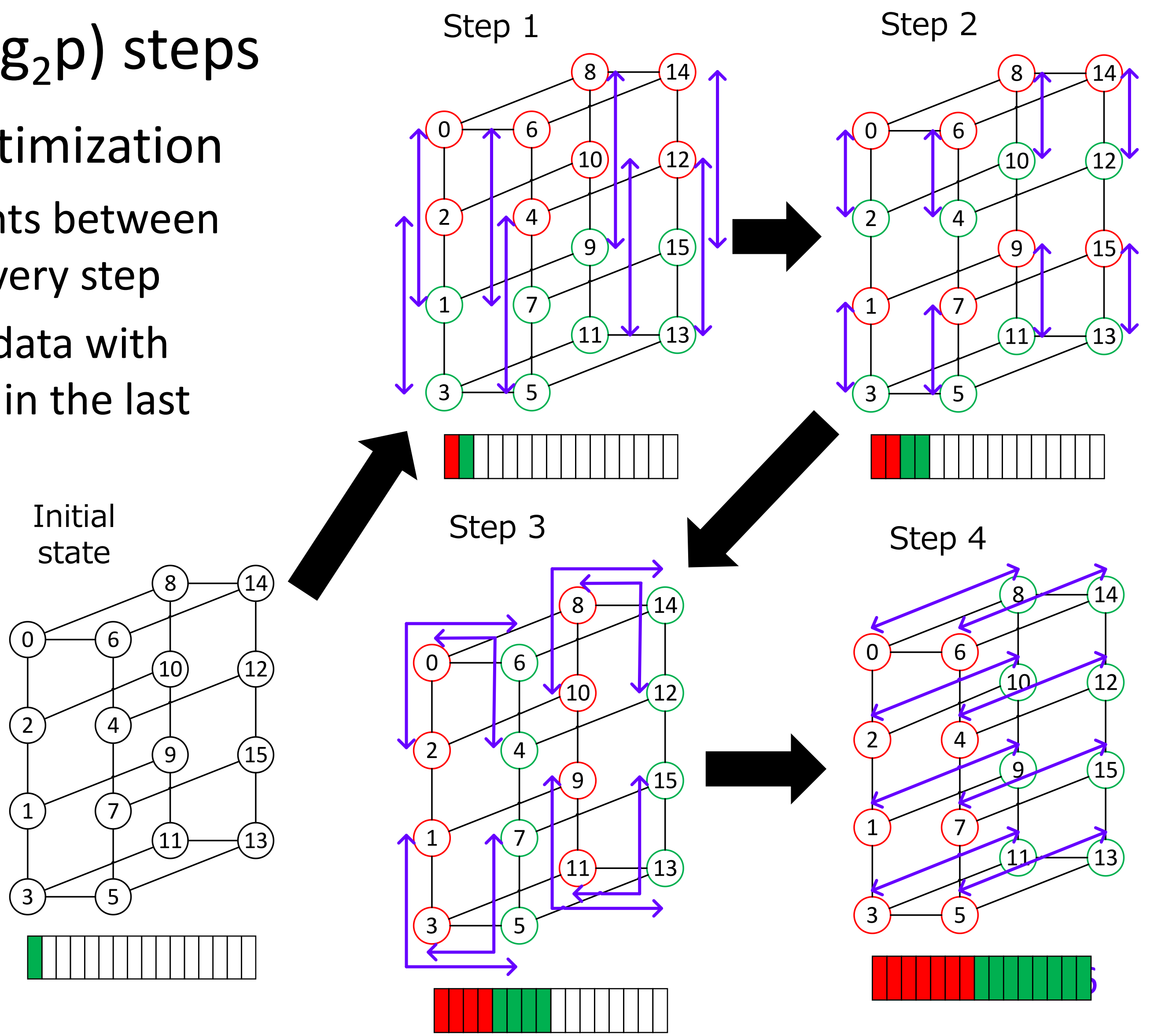
4. Dissemination algo.

Communication steps: $\log_2 p$

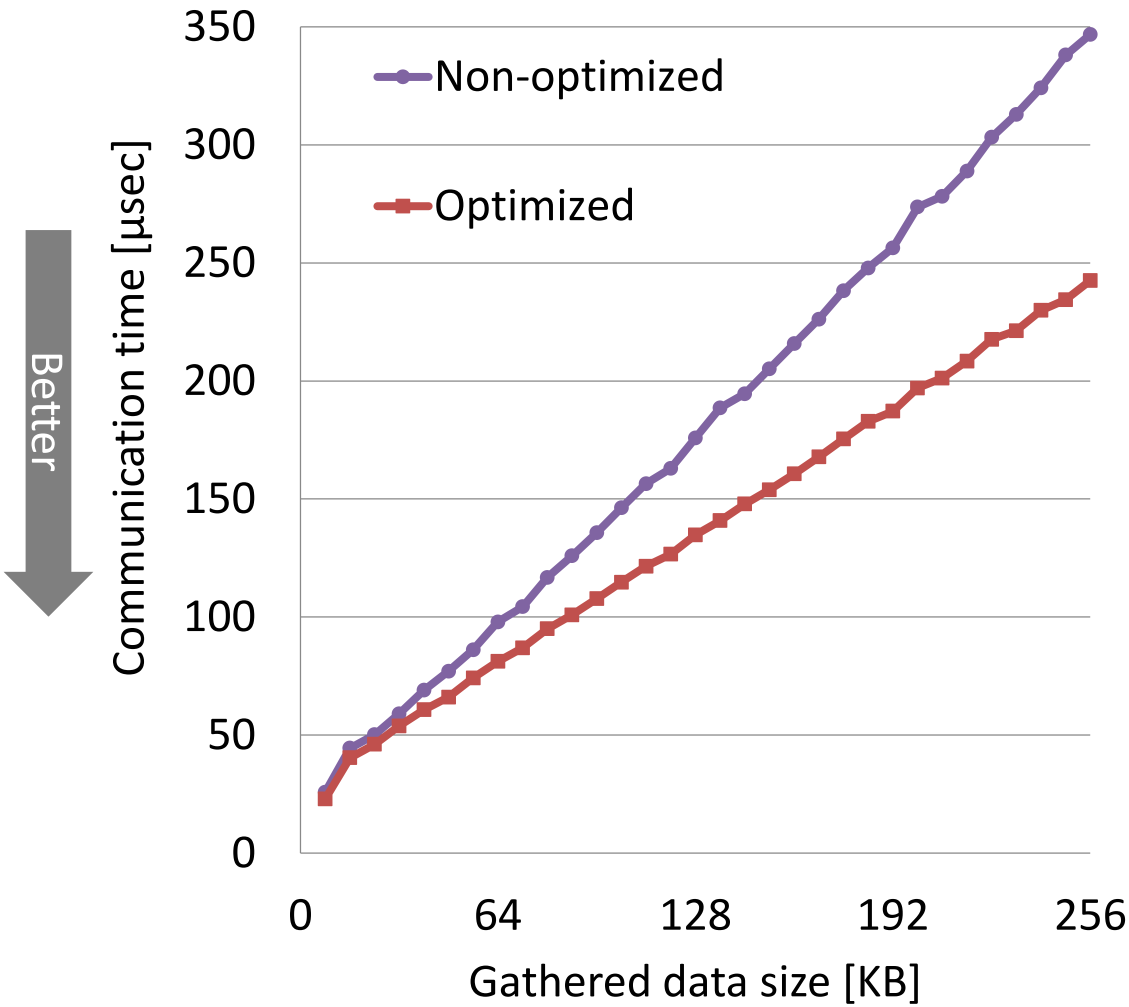


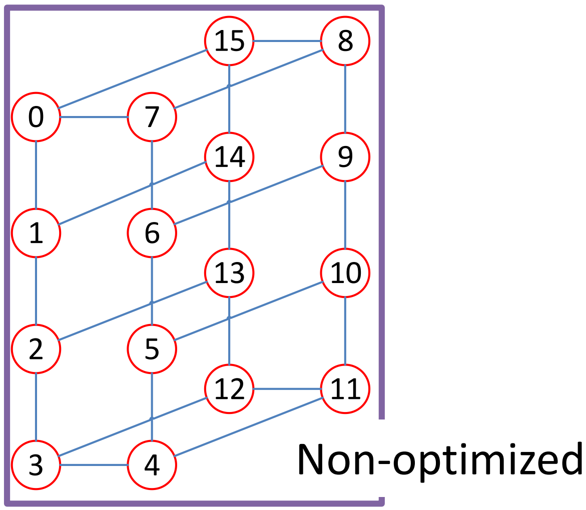
Allgather Implementation: Recursive Doubling (In case #processes = 16)

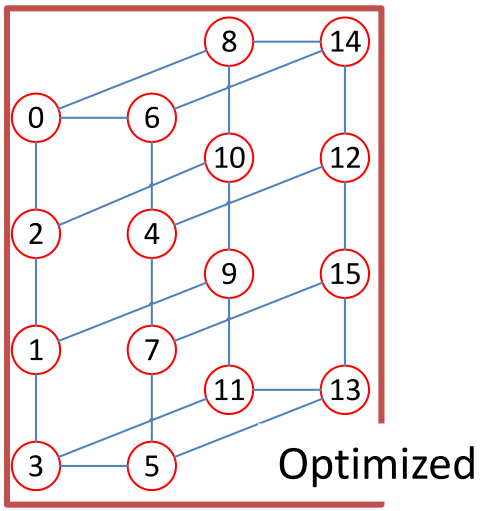
- Requires 4 (= $\log_2 p$) steps
- Node mapping optimization
  1. Same hop counts between any nodes in every step
  2. Communicate data with neighbor node in the last step



Impact of Node Mapping to Allgather Performance (#Processes=16)

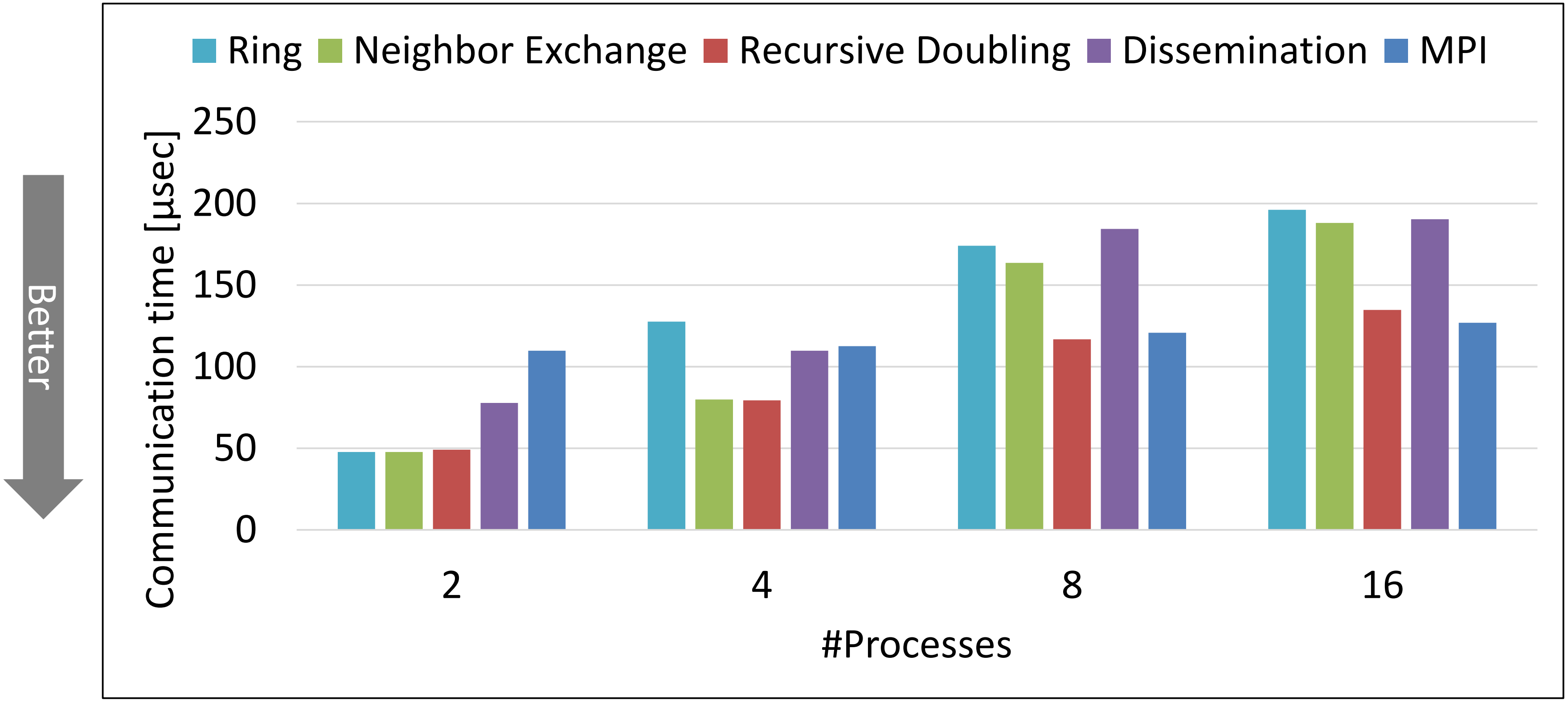




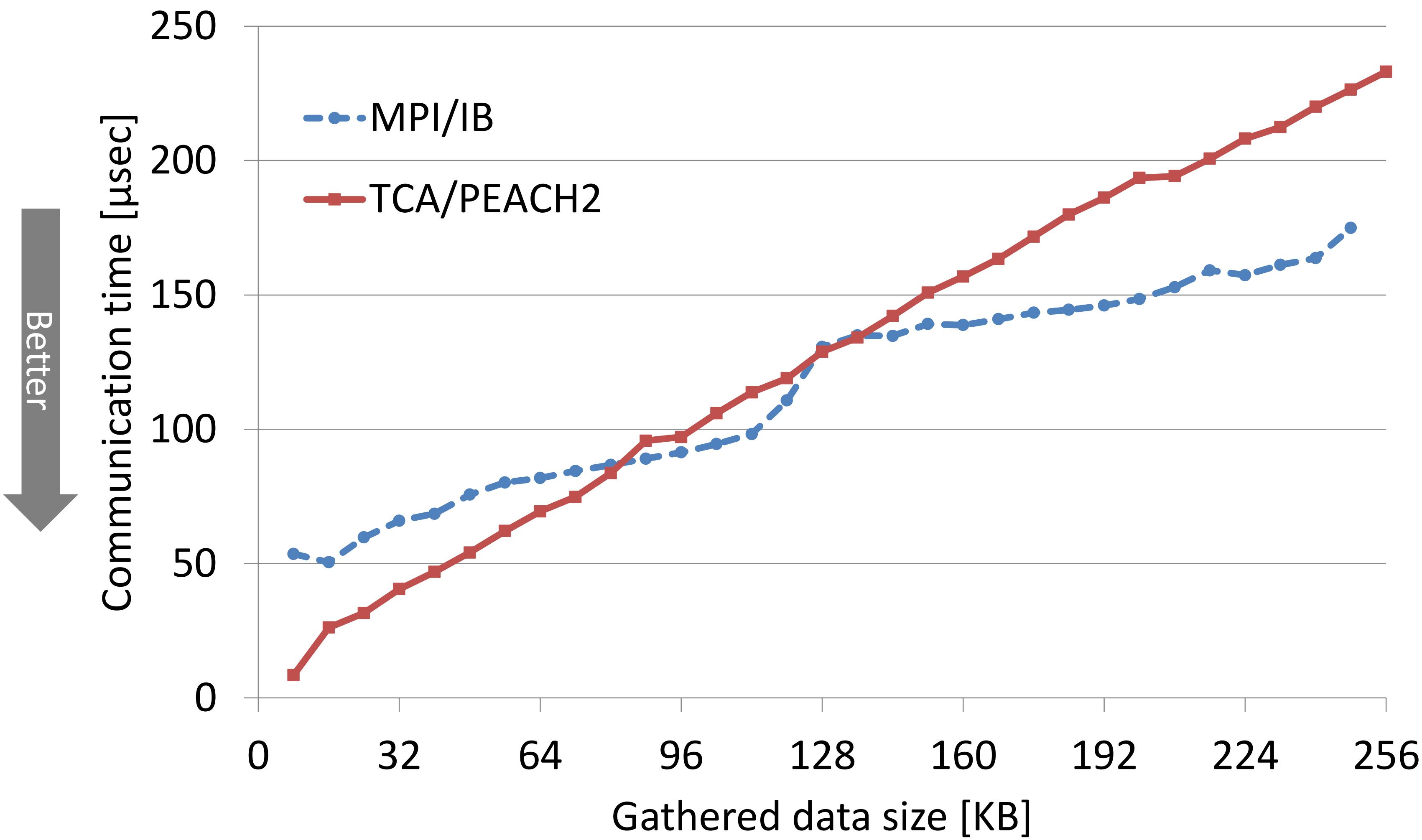


Allgather Perfomance Comparison among Different Algorithms

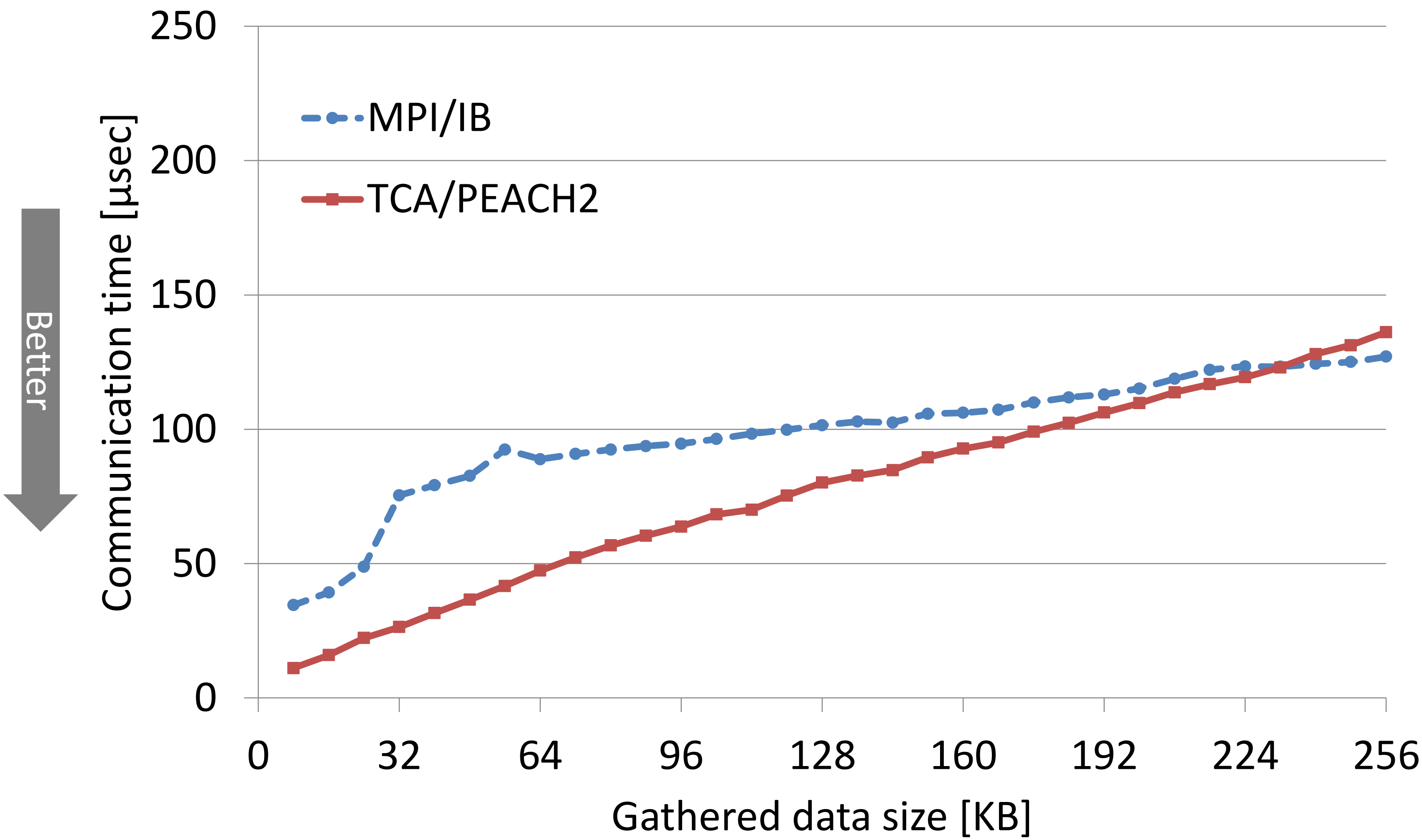
- Time for all-gathering **128 KB** data
  - N=16384 case in CG method
- **Recursive Doubling shows good performance**
  - However, when p=16, TCA is slower than MPI for this size



Allgather Performance (#Processes=16)

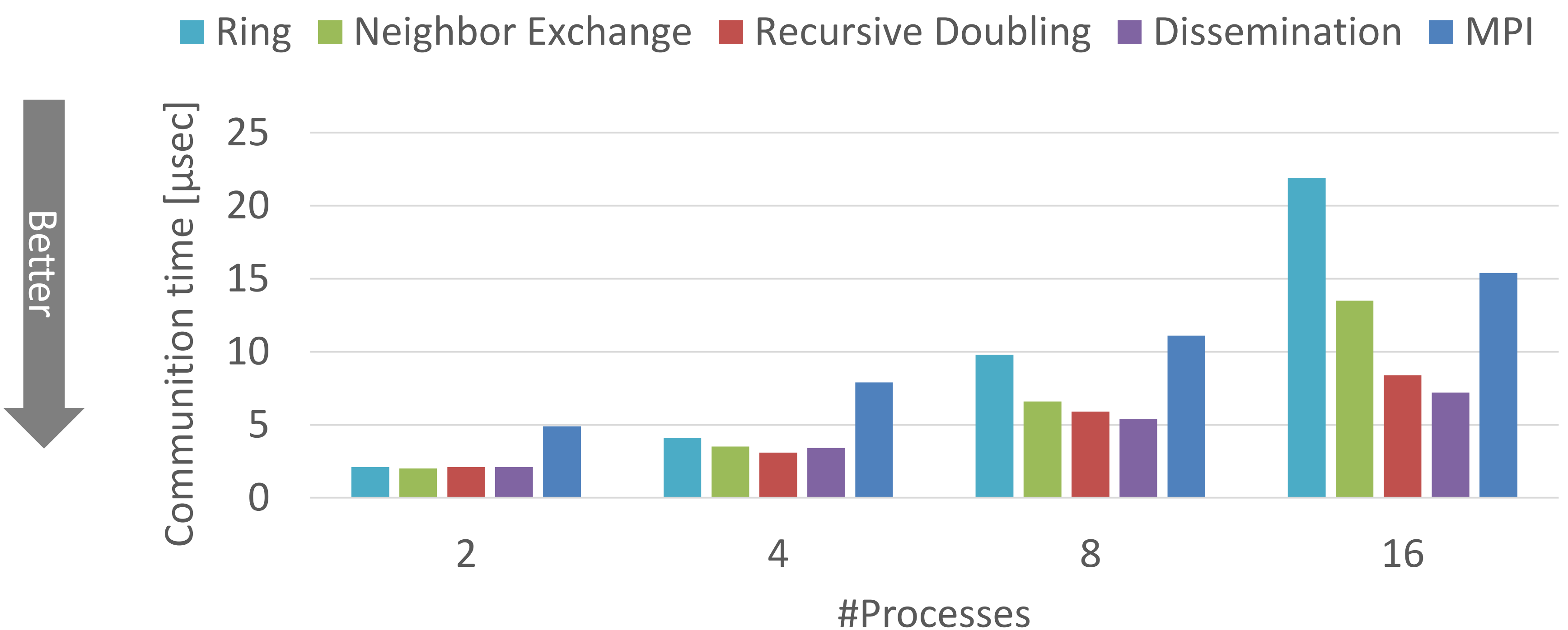


Allgather Performance (#Processes=4)



Allreduce Performance

- CPU-to-CPU allreduce time for 8 Bytes scalar data
- **Dissemination algorithm is the fastest.**
- TCA/PEACH2 is **more than 2x faster** than MPI/IB
  - Low latency of TCA works effectively



Outline

- Background and Motivation
- TCA (Tightly Coupled Accelerators) Architecture
- Collective Communication
  - Allgather and Allreduce
- **CG Method**
- Conclusion

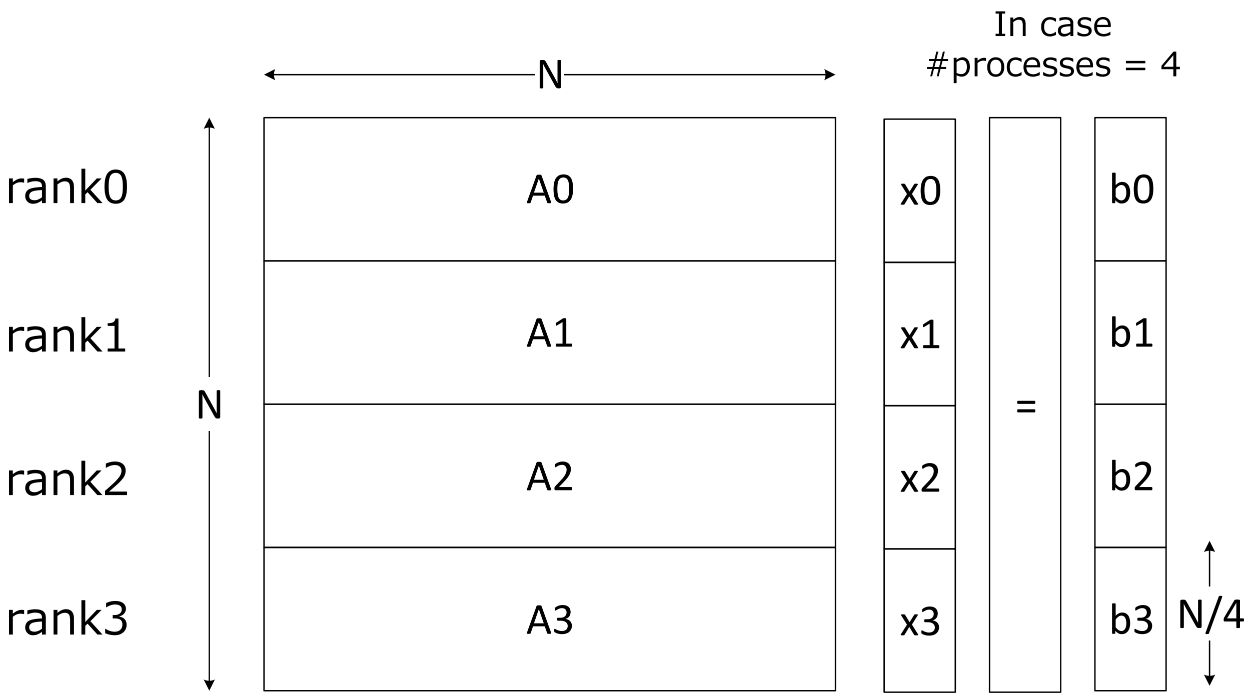
CG (Conjugate Gradient) Method

- Iterative solution for systems of linear equations
  - $Ax = b$
    - A: N-by-N symmetric positive-definite matrix (sparse matrix)
      - Sparse matrix is stored in **CRS (Compressed Row Storage)** order.
    - x, b: N-dimensional vector
  - No preprocessing
- Main computation parts (NVIDIA's **cuSPARSE** and **cuBLAS** are utilized)
  - **SpMV** x1 – Sparse Matrix-Vector Multiply ($\boldsymbol{q} := A\boldsymbol{p}$)
  - **DOT** x3 – Vector Dot Product ($\alpha := \boldsymbol{p}^T\boldsymbol{q}$)
  - **AXPY** x3 – Vector Multiply-Add ($\boldsymbol{y} := \alpha\boldsymbol{x} + \boldsymbol{y}$)

$$
\begin{array}{l}
\boldsymbol{r} := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \\
norm0 := \mathrm{sqrt}(\boldsymbol{r}^T\boldsymbol{r}) \\
\textbf{for } k := 1, 2, \cdots \textbf{ do} \\
\quad \rho := \boxed{\boldsymbol{r}^T\boldsymbol{r}} \\
\quad \textbf{if } k = 1 \textbf{ then} \\
\quad\quad \boldsymbol{p} := \boldsymbol{r} \\
\quad \textbf{else} \\
\quad\quad \beta := \rho/\rho\mathrm{prev} \\
\quad\quad \boldsymbol{p} := \beta\boxed{\boldsymbol{p} + \boldsymbol{r}} \\
\quad \textbf{end if} \\
\quad \boldsymbol{q} := \boxed{\boldsymbol{A}\boldsymbol{p}} \\
\quad \alpha := \rho/\boxed{(\boldsymbol{p}^T\boldsymbol{q})} \\
\quad \boldsymbol{x} := \boxed{\alpha\boldsymbol{p} + \boldsymbol{x}} \\
\quad \boldsymbol{r} := \boxed{-\alpha\boldsymbol{q} + \boldsymbol{r}} \\
\quad norm := \mathrm{sqrt}\boxed{(\boldsymbol{r}^T\boldsymbol{r})} \\
\quad \textbf{if } norm/norm0 < \varepsilon \textbf{ then} \\
\quad\quad \mathrm{break} \\
\quad \textbf{end if} \\
\quad \rho\mathrm{prev} := \rho \\
\textbf{end for}
\end{array}
$$

Parallelization of CG Method

- Parallelized by **row-wise one-dimensional partitioning** of matrix A



$\boldsymbol{x} := \mathrm{Allgather}(\boldsymbol{x}_l)$
$\boldsymbol{r}_l := \boldsymbol{b}_l - \boldsymbol{A}_l\boldsymbol{x}$
$d_t := \boldsymbol{r}_l^T\boldsymbol{r}_l$
$norm0 := \mathrm{sqrt}(\mathrm{AllreduceSum}(d_t))$
for $k := 1, 2, \cdots$ **do**
  $\rho_t := \boldsymbol{r}_l^T\boldsymbol{r}_l$
  $\rho := \boxed{\mathrm{AllreduceSum}(\rho_t)}$
  if $k = 1$ **then**
    $\boldsymbol{p}_l := \boldsymbol{r}_l$
  else
    $\beta := \rho/\rho\mathrm{prev}$
    $\boldsymbol{p}_l := \beta\boldsymbol{p}_l + \boldsymbol{r}_l$
  end if
  $\boldsymbol{p} := \boxed{\mathrm{Allgather}(\boldsymbol{p}_l)}$
  $\boldsymbol{q}_l := \boldsymbol{A}_l\boldsymbol{p}$
  $\alpha_t := \rho/(\boldsymbol{p}_l^T\boldsymbol{q}_l)$
  $\alpha := \boxed{\mathrm{AllreduceSum}(\alpha_t)}$
  $\boldsymbol{x}_l := \alpha\boldsymbol{p}_l + \boldsymbol{x}_l$
  $\boldsymbol{r}_l := -\alpha\boldsymbol{q}_l + \boldsymbol{r}_l$
  $d_t := \boldsymbol{r}_l^T\boldsymbol{r}_l$
  $norm := \mathrm{sqrt}(\boxed{\mathrm{AllreduceSum}(d_t)})$
  if $norm/norm0 < \varepsilon$ **then**
    break
  end if
  $\rho\mathrm{prev} := \rho$
end for

Parallelization of CG Method

- Parallelized CG method requires **collective communications** among all processes
  1. **Allgather** : Gathering required vector data for SpMV
  2. **Allreduce**: Reduction for having the summation of the local dot product
- Implemented collective communications are utilized.

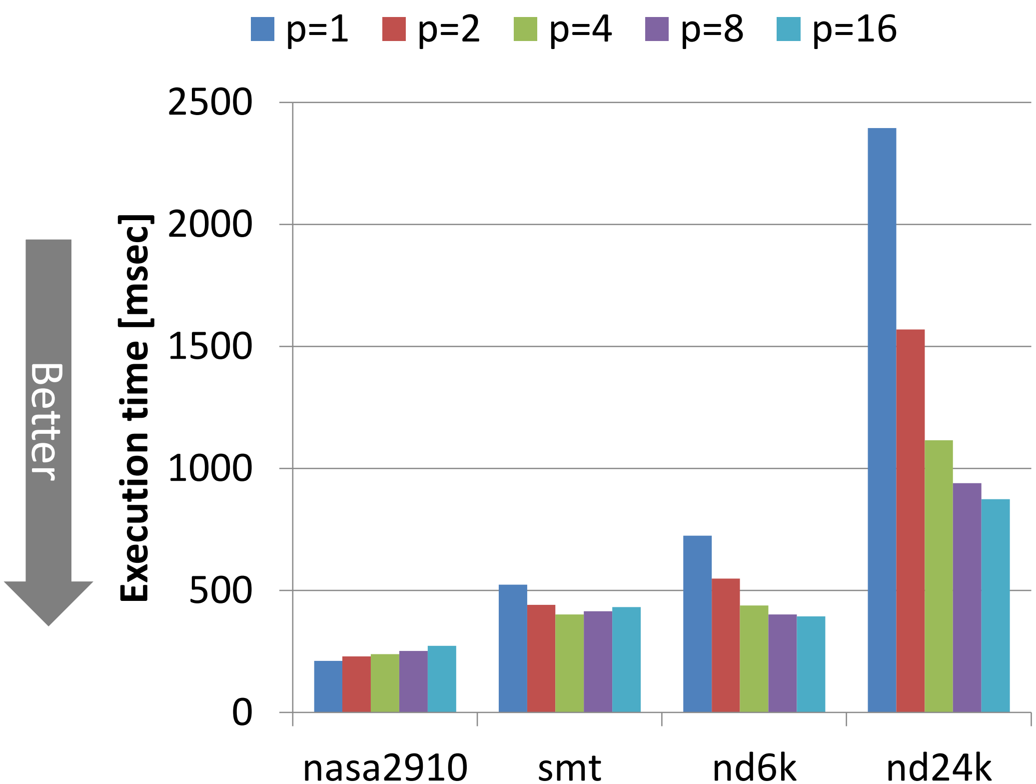
$$
\begin{aligned}
&\boldsymbol{x} := \mathrm{Allgather}(\boldsymbol{x}_l)\\
&\boldsymbol{r}_l := \boldsymbol{b}_l - \boldsymbol{A}_l\boldsymbol{x}\\
&d_t := \boldsymbol{r}_l^T\boldsymbol{r}_l\\
&norm0 := \mathrm{sqrt}(\mathrm{AllreduceSum}(d_t))\\
&\textbf{for } k := 1, 2, \cdots \textbf{ do}\\
&\quad \rho_t := \boldsymbol{r}_l^T\boldsymbol{r}_l\\
&\quad \rho := \boxed{\mathrm{AllreduceSum}(\rho_t)}\\
&\quad \textbf{if } k = 1 \textbf{ then}\\
&\quad\quad \boldsymbol{p}_l := \boldsymbol{r}_l\\
&\quad \textbf{else}\\
&\quad\quad \beta := \rho/\rho\mathrm{prev}\\
&\quad\quad \boldsymbol{p}_l := \beta\boldsymbol{p}_l + \boldsymbol{r}_l\\
&\quad \textbf{end if}\\
&\quad \boldsymbol{p} := \boxed{\mathrm{Allgather}(\boldsymbol{p}_l)}\\
&\quad \boldsymbol{q}_l := \boldsymbol{A}_l\boldsymbol{p}\\
&\quad \alpha_t := \rho/(\boldsymbol{p}_l^T\boldsymbol{q}_l)\\
&\quad \alpha := \boxed{\mathrm{AllreduceSum}(\alpha_t)}\\
&\quad \boldsymbol{x}_l := \alpha\boldsymbol{p}_l + \boldsymbol{x}_l\\
&\quad \boldsymbol{r}_l := -\alpha\boldsymbol{q}_l + \boldsymbol{r}_l\\
&\quad d_t := \boldsymbol{r}_l^T\boldsymbol{r}_l\\
&\quad norm := \mathrm{sqrt}(\boxed{\mathrm{AllreduceSum}(d_t)})\\
&\quad \textbf{if } norm/norm0 < \varepsilon \textbf{ then}\\
&\quad\quad \mathrm{break}\\
&\quad \textbf{end if}\\
&\quad \rho\mathrm{prev} := \rho\\
&\textbf{end for}
\end{aligned}
$$

CG Method Performance: Target Sparse Matrices

- Sparse matrices are from Univ. Florida Sparse Matrix Collection
- Matrix size (#Rows) is 1,000s to 10,000s

Matrix Name	nasa2910	smt	nd6k	nd24k
#Rows (N)	2,910	25,710	18,000	72,000
#Non-zero (nnz)	174,296	3,753,184	6,897,316	28,715,634
nnz/N	59.9	146.0	383.2	398.8

CG Method Performance



Matrix name	nasa2910	smt	nd6k	nd24k
#Rows (N)	2,910	25,710	18,000	72,000

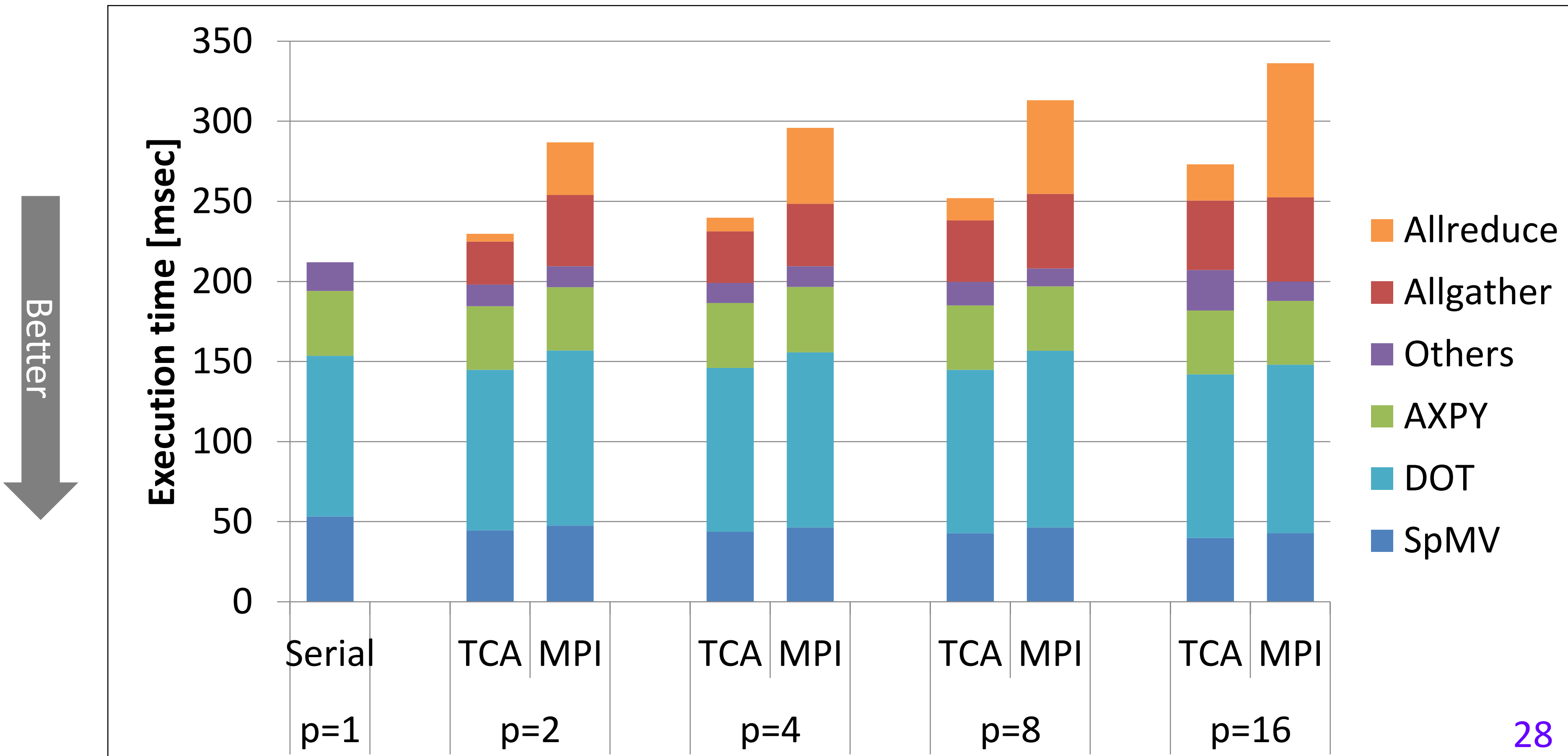
- Time for 1,000 iterations
  - Allgather is implemented with recursive doubling
  - Allreduce is implemented with dissemination algo.
- For **nd6k, nd24k**, parallelization yields **improvement**.
- For **smt**, using 4 processes is the fastest.
- For **nasa2910**, parallelization **deteriorates the performance**.

CG Method Performance: Time breakdown (nasa2910)

N=2,910, nnz=174,296

Breakdown of rank0

- **TCA is faster than MPI**, but performance does not scale

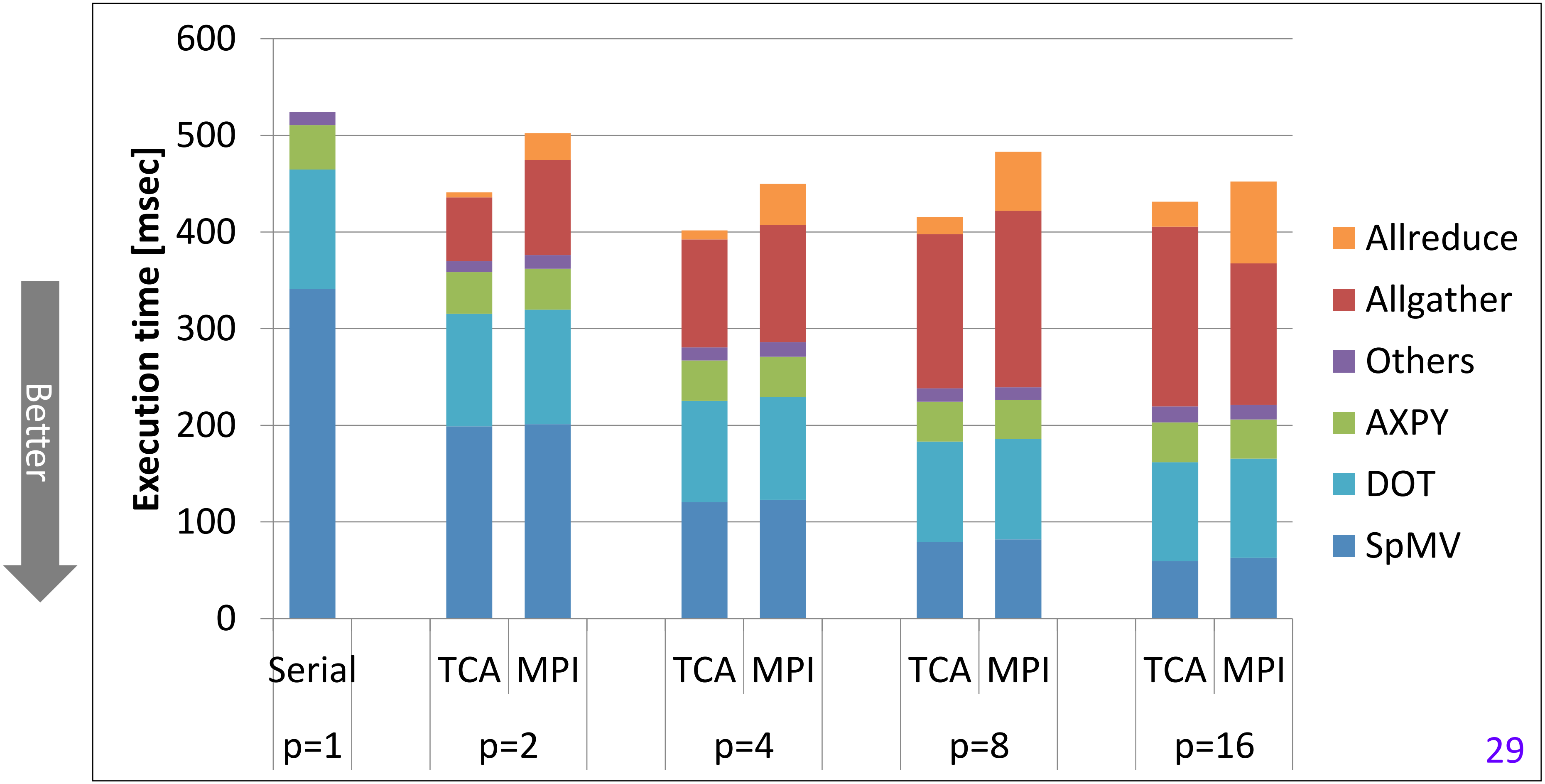


CG Method Performance: Time breakdown (smt)

N=25,710, nnz=3,753,184

Breakdown of rank0

- **TCA improves the performance.**

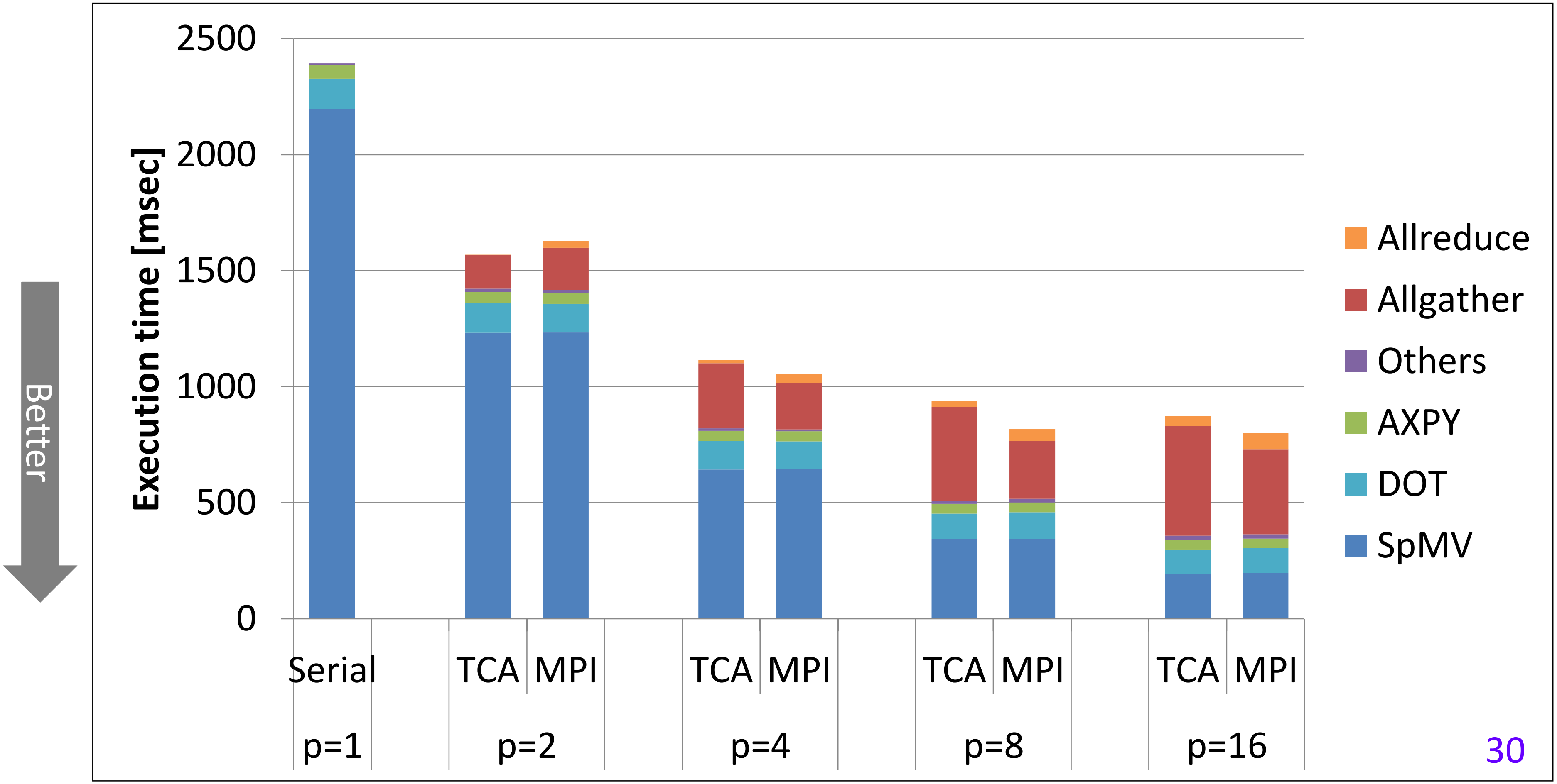


CG Method Performance: Time breakdown (**nd24k**)

N=72,000, nnz=28,715,634

Breakdown of rank0

- Performance scale well, but **TCA is not faster than MPI.**



Discussion

Matrix size	Small (1,000s)	Medium (10,000s)	Large
Performance improvement by TCA	**Large**	**Not-so-bad**	**No**
Strong scalability	**No**	**Not-so-bad**	**High**

- Implementing CG method with one-dimensional partitioning is not very suitable for TCA utilization.
  - We plan to implement and evaluate CG method with two dimensional partitioning.

Conclusion

- Collective communication using TCA/PEACH2's low latency communication improves the performance for small sizes.
- TCA improves the performance of CG method under specific conditions (10,000s rows of matrix).
- We will continue the research on TCA
- Future work:
  Making performance models to predict impact of TCA utilization to the performance.